Serijalizacija i deserijalizacija

Serijalizacija u programiranju predstavlja čin konvertovanja strukturiranih podataka iz aplikacije u obrazac u formatu koji kasnije omogućava oporavak njihove originalne strukture. Ovaj proces naziva se još i renderovanje podataka. Drugim rečima, pokretanjem procesa serijalizacije kreira se neki obrazac podataka prema zadatoj šemi.

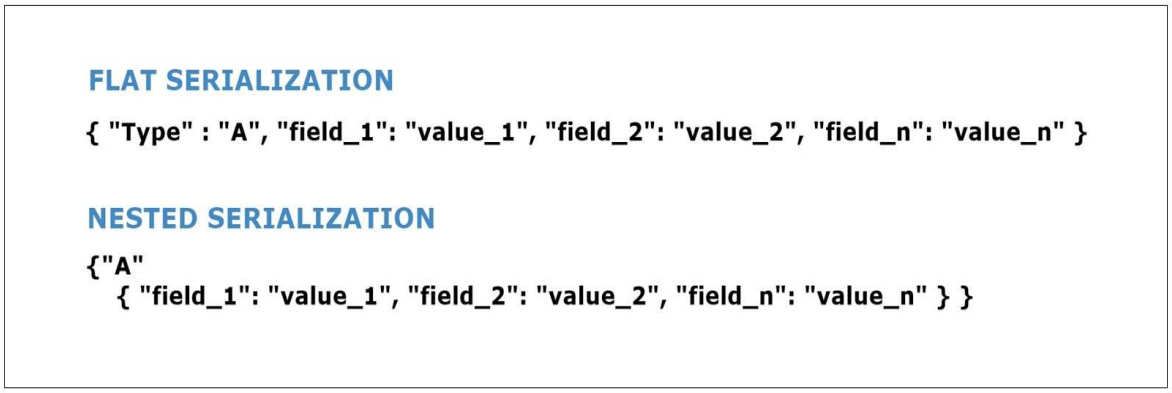
U računarstvu, osnovna jedinica za skladištenje informacija naziva se čvor. Čvorovi predstavljaju individualne delove neke veće strukture kao što su liste ili stabla, o kojima će biti više reči u narednoj lekciji. U procesu serijalizacije, za svaki čvor u šemi koji je zadat u aplikaciji kreira se po jedno polje. Ovaj proces se ponavlja rekurzivno dok sva polja ne budu kreirana. Kao rezultat rekurzije formira se stablo polja koje predstavlja preslikanu šemu. Svaki objekat polja nastao kao rezultat prethodnog koraka u sebi ima atribut koji indikuje sa kojim čvorom u šemi je povezan.

Deserijalizacija je proces suprotan serijalizaciji, koji omogućava rekonstrukciju originalnog objekta. Ona obrazac konvertuje u podatke koji se prikazuju na aplikaciji. Postupak je identičan postupku serijalizacije, odnosno preslikavanju čvorova u polja rekurzivno.

Pre nego što započnemo proces serijalizacije, važno je da odredimo kako će serijalizovani podaci biti strukturirani. Prema ovom kriterijumu, postoje dva tipa serijalizacije:

- **ravna** – tekstualni fajlovi, CSV fajlovi, NumPy nizovi;
- **ugnežđena** – Pickle, JSON, XML, YAML.

Razlika u strukturi ova dva tipa prikazana je na slici 2.1.



Slika 2.1. Opšti prikaz ravne i ugnežđene serijalizacije u kodu

Najpopularniji ugrađeni modul za serijalizaciju podataka u Python programskom jeziku je **`pickle`**.

Modul pickle

Karakteristika ovog modula je da je on Python-specifičan. To znači da programi koji nisu pisani u jeziku Python mogu imati problem sa deserijalizacijom podataka koji su serijalizovani korišćenjem ovog modula. Još jedna od karakteristika `pickle` modula je da njegov format koristi relativno kompaktan binarni prikaz i ima mogućnost kompresovanja podataka. Da bismo mogli da upotrebljavamo funkcije ovog modula, neophodno je da ga uvedemo u program, što činimo komandom `import pickle`.

Dakle, *picklovanje* služi za konverziju objekata u binarni kod i njihovo vraćanje iz binarnog koda u prvobitno stanje. Ovakva konstrukcija i dekonstrukcija Python objekata vrši se u skladu sa definisanim protokolima (tabela 2.1).

Protokoli serijalizacije objekata u `pickle` modulu	
Protokol	**Opis**
protokol verzije 0	originalni „ljudski čitljiv" protokol
protokol verzije 1	stari binarni format kompatibilan sa starijim verzijama Pythona
protokol verzije 2	uveden u Pythonu 2.3; omogućava efikasnu serijalizaciju novih klasa
protokol verzije 3	uveden u Pythonu 3.0; kompatibilan sa drugim Python 3 verzijama
protokol verzije 4	uveden u Python 3.4; podržava skladištenje velikih objekata

Tabela 2.1. Protokoli serijalizacije objekata u pickle modulu

Kako bismo proverili koji je najviši protokol, kao i koji je podrazumevani protokol koji naša verzija Pythona koristi, pozvaćemo ugrađene modul funkcije `pickle.HIGHEST_PROTOCOL` i `pickle.DEFAULT_PROTOCOL`.

Primer upotrebe funkcija za proveru protokola u *pickle* modulu

```
import pickle
print(pickle.HIGHEST_PROTOCOL)
print(pickle.DEFAULT_PROTOCOL)
```

Dve osnovne funkcije koje obavlja `pickle` interfejs su:

- funkcija serijalizacije:
  - `dump()` – serijalizacija podataka u objekte;
  - `dumps()` – serijalizacija podataka u string;
- funkcija deserijalizacije:
  - `load()` – deserijalizacija podataka u objekte;
  - `loads()` – deserijalizacija podataka u string.

Kada vršimo serijalizaciju podataka u objekte, koristimo funkciju **`dump()`**. Ova funkcija ima dva obavezna ulazna parametra. Prvi parametar predstavlja podatke koje želimo da serijalizujemo, dok drugi predstavlja funkciju koja te podatke upisuje u fajl.

Da bismo imali realnu sliku o tome kako se serijalizacija odvija, definisaćemo jednu n-torku podataka koju ćemo kasnije upisati. Korišćenjem built-in funkcije `open()`, čija je uloga opisana u ranijim lekcijama, definisaćemo fajl `books_file` u koji će se podaci upisivati kao objekti. Konačno, upotrebom `dump()` funkcije prosleđujemo podatke `books` koje želimo da serijalizujemo i parametar `books_pickled`, koji objekte upisuje u željeni fajl.

Primer upotrebe `dump()` funkcije u `pickle` modulu

```
import pickle

books = ('J.K. Rowling', 'Harry Potter',
          ((1997, 'Harry Potter and the Philosopher\'s Stone'),
          (1998, 'Harry Potter and the Chamber of Secrets'),
          (1999, 'Harry Potter and the Prisoner of Azkaban'),
          (2000, 'Harry Potter and the Goblet of Fire'),
          (2003, 'Harry Potter and the Order of the Phoenix'),
          (2005, 'Harry Potter and the Half-Blood Prince'),
          (2007, 'Harry Potter and the Deathly Hallows')))

with open('books_file', 'wb') as books_pickled:
    pickle.dump(books, books_pickled)
```

Funkcija `pickle` modula **`dumps()`** koristi se na sličan način. Ova funkcija, za razliku od `dump()` ima samo jedan obavezan parametar, i to su podaci za serijalizaciju. Korišćenjem ove funkcije podaci se ne upisuju u fajl u vidu objekata, već se skladište u formi stringova. Da bismo prikazali formu skladištenja, rezultat funkcije `dumps()` ispisaćemo na ekranu.

Primer upotrebe `dumps()` funkcije u `pickle` modulu

```
import pickle

books = ('J.K. Rowling', 'Harry Potter',
          ((1997, 'Harry Potter and the Philosopher\'s Stone'),
          (1998, 'Harry Potter and the Chamber of Secrets'),
          (1999, 'Harry Potter and the Prisoner of Azkaban'),
          (2000, 'Harry Potter and the Goblet of Fire'),
          (2003, 'Harry Potter and the Order of the Phoenix'),
          (2005, 'Harry Potter and the Half-Blood Prince'),
          (2007, 'Harry Potter and the Deathly Hallows')))

print(pickle.dumps(books))
```

```
b"\x80\x04\x95i\x01\x00\x00\x00\x00\x00\x8c\x0cJ.K. Rowling\x94\x8c\x0cHarry Potter
\x94(M\xcd\x07\x8c(Harry Potter and the Philosopher's Stone\x94\x86
\x94M\xce\x07\x8c'Harry Potter and the Chamber of Secrets\x94\x86
\x94M\xcf\x07\x8c(Harry Potter and the Prisoner of Azkaban\x94\x86
\x94M\xd0\x07\x8c#Harry Potter and the Goblet of Fire\x94\x86
\x94M\xd3\x07\x8c)Harry Potter and the Order of the Phoenix\x94\x86
\x94M\xd5\x07\x8c&Harry Potter and the Half-Blood Prince\x94\x86
\x94M\xd7\x07\x8c$Harry Potter and the Deathly Hallows\x94\x86\x94t\x94\x87\x94."
```

Slika 2.2. Prikaz forme skladištenja podataka korišćenjem dumps() funkcije

Kao što možemo da primetimo, serijalizacija podataka čini da podaci koji su predmet serijalizacije menjaju svoju formu pri skladištenju. Iz ovog razloga, kada te podatke želimo da deserijalizujemo, odnosno vratimo u originalan oblik, to moramo učiniti pomoću funkcije `load()`, odnosno `loads()`.

Funkcija **`load()`** koristi se za učitavanje podataka koji su na fajlovima skladišteni u formi objekata. Učitavanje podataka sa fajla vršimo pomoću funkcije `open()`, korišćenjem moda za čitanje binarnog zapisa. Ovako definisanu funkciju prosleđujemo funkciji `load()` u vidu parametra, kako bi se učitani podaci deserijalizovali.

Primer upotrebe `load()` funkcije u `pickle` modulu

```
with open('books_file', 'rb') as books_unpickled:
    pickle.load(books_unpickled)
```

Za deserijalizaciju podataka koji su skladišteni u formi stringova koristimo **`loads()`**. Ova funkcija kao parametar prima string podatke u formi koja je kreirana korišćenjem `dumps()` funkcije i vraća ih u originalni oblik.

Primer upotrebe `loads()` funkcije u `pickle` modulu

```
pickle.loads(b"\x80\x04\x95i\x01\x00\x00\x00\x00\x00\x00\x8c\x0cJ.K.
Rowling\x94\x8c\x0cHarry Potter\x94(M\xcd\x07\x8c(Harry Potter and the
Philosopher's Stone\x94\x86\x94M\xce\x07\x8c'Harry Potter and the
Chamber of Secrets\x94\x86\x94M\xcf\x07\x8c(Harry Potter and the
Prisoner of Azkaban\x94\x86\x94M\xd0\x07\x8c#Harry Potter and the
Goblet of Fire\x94\x86\x94M\xd3\x07\x8c)Harry Potter and the Order of
the Phoenix\x94\x86\x94M\xd5\x07\x8c&Harry Potter and the Half-Blood
Prince\x94\x86\x94M\xd7\x07\x8c$Harry Potter and the Deathly
Hallows\x94\x86\x94t\x94\x87\x94.")
```

Izmenite primer tako da se umesto filmova serijalizuju Vaše omiljene knjige u fajlu books pa potom i deserijalizuju.

Modul marshal

Modul **`marshal`** takođe služi za čitanje i pisanje podataka u binarnom formatu. Za razliku od `pickle`, koji vrši praćenje serijalizovanih objekata dodelom referenci na njih, `marshal` se ne može koristiti za serijalizaciju korisnički definisanih podataka.

Ovaj modul uglavnom postoji za čitanje i pisanje pseudokompajliranog koda za Python module iz `.pyc` datoteka. Za njegovo korišćenje neophodno je uvođenje modula u program, što činimo komandom `import marshal`.

Napomena

Pseudokompajlirani kod predstavlja kod koji je napisan u naprednom jeziku, odnosno, zahteva dalje kompajliranje da bi mogao da se izvrši.

Za serijalizaciju i deserijalizaciju podataka u `marshal` modulu koristimo identične funkcije kao i u `pickle` modulu:

- funkcija serijalizacije:
    - `dump()` – serijalizacija podataka u objekte;
    - `dumps()` – serijalizacija podataka u string;

- funkcija deserijalizacije:
    - `load()` – deserijalizacija podataka u objekte;
    - `loads()` – deserijalizacija podataka u string.

Ove funkcije rade po istom principu i koriste iste parametre kao i funkcije `pickle` modula, te ih nećemo detaljno objašnjavati.

Pitanje

Za serijalizaciju podataka u formi objekata koristimo funkciju:

- load();
- **dump();**
- dumps();
- loads();

Objašnjenje:

Za serijalizaciju podataka u formi objekata koristimo funkciju dump(). Funkcija dumps() služi za serijalizaciju podataka u formi stringova, dok load() i loads() služe za deserijalizaciju.

Modul shelve

Modul **`shelve`** služi za serijalizaciju i deserijalizaciju podataka u formatu rečnika, zbog čega predstavlja odličan alat za upis podataka u bazu. Kao što smo mogli da primetimo u lekciji o tipovima podataka, rečnik skladišti podatke u parovima ključ–vrednost. Ono što `shelve` modul omogućava je da vrednost u tom paru bude objekat, dok ključ tog para mora biti običan string.

Za korišćenje `shelve` modula neophodno je njegovo uvođenje u program, što ćemo učiniti linijom `import shelve`. Sada kada imamo pristup funkcijama ovog modula, kreiraćemo fajl `ShelveExample`, u koji ćemo upisati neke ključ–vrednost podatke. Podaci se u ovom modulu upisuju pomoću funkcije `open()`, koja kao parametar prima ime fajla u koji vršimo upis. U okviru funkcije definišemo ključ–vrednost podatke. Pogledajmo primer upotrebe open() funkcije u shelve modulu:

Radno okruženje

```
import shelve

with shelve.open('ShelveExample') as mutd_jerseys:
    mutd_jerseys['1'] = 'David de Gea'
    mutd_jerseys['2'] = 'Victor Lindelof'
    mutd_jerseys['3'] = 'Eric Bailly'
    mutd_jerseys['4'] = 'Phil Jones'
    mutd_jerseys['5'] = 'Harry Maguire'
    mutd_jerseys['6'] = 'Paul Pogba'
    mutd_jerseys['7'] = 'Alexis Sanchez'
    mutd_jerseys['8'] = 'Juan Mata'

    for key in mutd_jerseys:
        print(key,' : ',mutd_jerseys[key])
```

Objašnjenje:

Ovako kreiran fajl predstavlja osnovu za bazu podataka koja ispod njega leži. Kao nuspojava može se desiti da program kreira više fajlova sa istim nazivom i različitim ekstenzijama. Fajl kreiran za bazu se može koristiti za čitanje i upisivanje podataka.

Manipulaciju vrednostima u rečniku vršimo preko ključeva. Ukoliko pri ponovnom korišćenju `open()` funkcije za isti fajl promenimo vrednost u paru ključ–vrednost, u fajlu će se za zadati ključ upisati ta nova vrednost. Ključ `7` će u daljem primeru u našoj bazi `ShelveExample` dobiti novu vrednost *David Bechkam*.

Primer promene vrednosti zadatom ključu u `shelve` modulu

```
import shelve

with shelve.open('ShelveExample') as mutd_jerseys:
    mutd_jerseys['7'] = 'David Beckham'
```

Modul `shelve` podržava sve funkcije koje podržava i rečnik. U nastavku je dat primer korišćenja `del` kao jedne od funkcija rečnika. U zadatom primeru prikazano je kako ova funkcija briše podatak iz naše baze korišćenjem zadatog ključa.

Primer upotrebe `del` funkcije u `shelve` modulu

```
import shelve

mutd_jerseys = shelve.open('ShelveExample')
del mutd_jerseys['7']
```

Izmenite kod tako da se izbriše igrač pod definisanim ključem 1 kao i promeni ime i prezime igrača sa definisanim ključem 2.

Radno okruženje

Modul json

Jedan od najčešće korišćenih modula za serijalizaciju i deserijalizaciju podataka je **`json`** modul. Ovaj modul je doslovno inspirisan sintaksom jezika JavaScript, iako ne predstavlja njegov podskup.

Za razliku od ostalih modula, `json` se koristi za serijalizaciju podataka u tekstualnom formatu, a ne u binarnom. Prednost ovakve vrste serijalizacije jeste to što omogućava *ljudsku čitljivost*. Još jedna od prednosti `json`-a jeste njegova interportabilnost, odnosno mogućnost čitanja i u drugim programima, a ne samo onim napisanim u Pythonu. Ono što `json` ne podržava jesu korisnički definisane klase, odnosno, može se koristiti samo za serijalizaciju i deserijalizaciju predefinisanih tipova podataka.

Modul `json` koristi isti API (application programming interface) kao i ostali Python moduli za serijalizaciju i deserijalizaciju. Zbog svoje interportabilnosti, veoma se često koristi. Stoga će njegova upotreba biti detaljno opisana u kursu *Service Application Development*.

Napomena

Nijedan od navedenih modula nije predviđen za to da pruža zaštitu od zlonamerno izgrađenih podataka. Preporuka je da se podaci iz nepouzdanih i neovlašćenih izvora ne deserijalizuju.

Rezime

- Serijalizacija u programiranju predstavlja čin konvertovanja strukturiranih podataka iz aplikacije u obrazac u formatu koji kasnije omogućava oporavak njihove originalne strukture.
- Deserijalizacija je proces koji omogućava rekonstrukciju originalnog objekta iz serijalizovanog obrasca.
- Dva osnovna tipa serijalizacije su ravna i ugnežđena; ovi tipovi razlikuju se u strukturi skladištenja.
- Najpopularniji modul serijalizacije je `pickle`, koji podatke serijalizuje u binarnom formatu. Konstrukcija i dekonstrukcija podataka vrše se po različitim protokolima čija podržanost zavisi od verzije Pythona. Osnovne funkcije `pickle` modula su:
    - funkcija serijalizacije:
        - `dump()` – serijalizacija podataka u objekte;
        - `dumps()` – serijalizacija podataka u string;
    - funkcija deserijalizacije:
        - `load()` – deserijalizacija podataka u objekte;
        - `loads()` – deserijalizacija podataka u string.
- Modul `marshal` vrši serijalizaciju podataka u binarnom formatu uz dodavanje referenci svakom od objekata. Uglavnom postoji za čitanje i pisanje pseudokompajliranog koda za Python module iz `.pyc` datoteka. Koristi iste funkcije kao `pickle`.

- Modul `shelve` služi za serijalizaciju i deserijalizaciju podataka u formatu rečnika. Predstavlja odličan alat za upis podataka u bazu. Omogućava da vrednost u paru ključ–vrednost bude objekat, dok ključ tog para mora biti običan string. Ima sve funkcije kao i rečnik. Manipulaciju podacima vrši preko zadatog ključa.
- Modul `json` koristi se za serijalizaciju i deserijalizaciju u tekstualnom formatu. Za njega su karakteristični ljudska čitljivost i interportabilnost. Koristi isti API kao i ostali moduli. Može da serijalizuje samo ugrađene tipove podataka, ne i korisničke definisane.